The R in gRaphics (CDT-32)

Luciano da Fontoura Costa
luciano@ifsc.usp.br

São Carlos Institute of Physics – DFCM/USP

28th Apr 2020

Abstract

Some of the many graphical resources available in R and related libraries are briefly presented and illustrated, including functions for drawing 2D curve plots, barplots and histograms, 2D and 3D scatterplots, images, 3D surfaces, 3D parametric surfaces, level-sets, and vector fields.

"The whole is greater than the part."

Book I, Euclid's Elements.

1 Introduction

R is a popular software environment in the physical sciences, especially in Statistics, though it can be applied in virtually every area. As an environment, R incorporates many native and library complemented resources and facilities ranging from numeric methods to complex network analysis. It can also provide great support for computer graphics and scientific visualization (e.g. [1, 2, 3]), which can be used in several ways including but by no means limited to: prototyping, teaching, art, modeling, debugging, and data analysis.

Graphics and visualization are so useful because they provide resources for perceiving jointly several characteristics and interrelationships in given data. The main limitation is that human perception is limited to three or four dimensions, but even higher dimensional spaces can be studied by using careful projections and slicing.

In order to complement and support the material covered in the CDT series [4], a concise introduction to some of the main features of the R environment was disseminated recently [5]. However, for simplicity's sake and so as to keep the length of that CDT reasonable, the many R-related resources for graphics and visualization were kept to a minimum. The present text provides an introduction to some of those concepts, therefore complementing the previous CDT to which it is related and can be understood as a continuation.

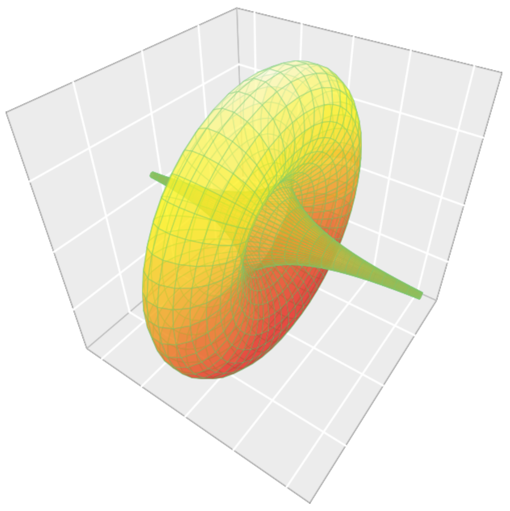


Figure 1: The *breather* surface is but one of the many examples of remarkable parametric surfaces that can be visualized in R.

In the present work, in addition to covering the basics of R graphics, we will delve into relatively more advanced topics such as parametric surfaces, level sets, and vector fields. However, as with the previous work, the presented material assumes a certain level of familiarity with areas such computer programming, linear algebra, and multivariate calculus (e.g. [6]). It is also observed that several aspects of the covered material can vary between machines, operating systems, versions and implementations. Also, there are many aspects that are not cover or only briefly mentioned here, and the reader needs to complement and double check the respective literature and reference material.

We start presenting some basic concept and progress to 2D plots, barplots and histograms, 3D scatterplots, images, meshes, surfaces, level sets, parametric surfaces, and vector fields.

2 Basic Graphics

In R, graphics are obtained through respective *devices*. For instance, the command ‘x11()’ followed by ‘dev.new()’ creates a new $x11$ device in a Unix-like environment, though it may work in other operating systems. Usually, a graphic device will be assigned by default a type that may depend on the current operating system. For instance, in the *Mac* platform, the default device will commonly be ‘quartz()’.

Other devices exist, including image formats, or printers. The external device type ‘svg()’ (vector based graphic, XML-related) may be considered when animations and interactivity are to be implemented. It is a good practice to close all devices once they are not used.

Devices of a same type can be managed independently. There are several commands that allow the specification and control of devices, such as ‘dev.new()’, for creating a new device; ‘dev.list()’, which lists the existing devices; ‘dev.cur()’ which goes back to the previous device respectively to that specified in its argument; ‘dev.off()’ which turns off a device; and ‘graphics.off()’ which turns all devices off. It is also possible to copy one device into another by using the ‘dev.copy()’ function.

The command ‘plot()’ can be used to generate $x \times y$ plots of curves. For instance, ‘plot(c(-1,1),c(-1,1)) will yield a solid diagonal line going from coordinates $[-1, -1]$ to $[1, 1]$. This command is further discussed in the subsequent section.

It is also possible to organize the screen of the current device as an array of *nrow* rows by *ncol* columns plots by using the command ‘par(mfrow=c(nrow,ncol))’. Each subsequent plot will act on one of the available successive canvas, in cycling fashion once all canvases have been activated.

In the present work, we will consider facilities that are native to R, such as the plot command, as well as facilities and resources provided by libraries. Table reftab:libr lists the main adopted commands derived from respective libraries. It should be kept in mind that, though sometimes having the same names, a directive can ben understood and have different effects when used in different commands or libraries. Also commands may be modified along time. The reader needs to double check the related references.

The R environment has several built-in datasets, a few

Table 1: Some of the R graphic functions and respective libraries.

graphic function	Library
plot3d()	rgl [7]
persp3d()	rgl [7]
scatter3d()	plot3d [8]
points3d()	plot3d [8]
arrows3d()	plot3d [8]
text3d()	plot3d [8]
surf3d()	plot3d [8]
mesh()	plot3d [8]
contour()	graphics [9]
quiver()	pracma [10]

of which will be used in the present text. The complete list of available datasets can be obtained through the command ‘data()’. Documentation about each R command can be obtained by using ‘help()’.

3 Two-Dimensional Plots

The basic native graphic command in R is arguably the ‘plot()’. Though operating only on 2D spaces, it has a wide range of options. A good understanding of how this command operates provides a good preparation for addressing other R more sophisticate R graphic commands, to be discussed in subsequent sections of the present work.

Basically, the function ‘plot(x,y)’ takes as argument two real-valued vectors ‘x’ and ‘y’ having the same length N, and then produces three main types of graphical output: (i) *separated points*, which is the default (type = ”p”); (ii) *polyline*, corresponding to the points joined by consecutive straight line segments, which needs the ‘type’ parameter to be set to line, i.e. ‘plot(x,y,type=”l”); and (iii) both, which can be achieved by ‘plot(x,y,type=”b”)’. Other interesting options include ‘plot(x,y,type=”o”)’ and ‘plot(x,y,type=”c”)’, which produce a continuous polyline with the points overlaid and a segmented polyline, respectively.

The following piece of code, whose graphical output is depicted in Figure 2, illustrates some of the various curve options in the plot command.

```
par(mai = c(0.3, 0.3, 0.3, 0.3), mfrow=c(4,1))
t <- seq(0, 2, 0.1);  y <- cos(2*pi*t)
plot(t,y,main="type p", type = "p")
plot(t,y,main="type l", type = "l")
plot(t,y,main="type c", type = "c")
plot(t,y,main="type b", type = "b")
plot(t,y,main="type o", type = "o")
```

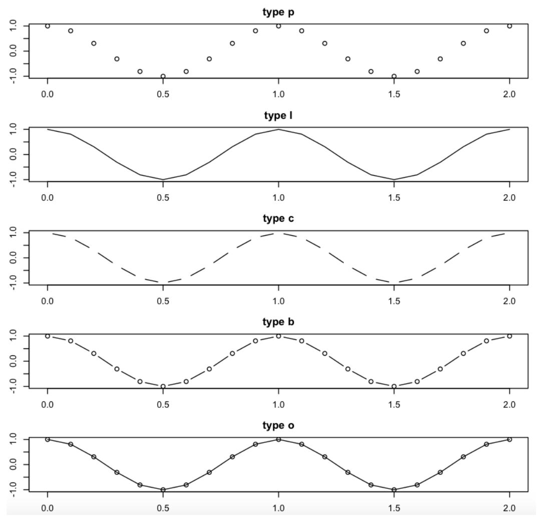


Figure 2: The graphic outputs obtained for a cosine function by using several types of curve plots.

The plot command can work jointly with the ‘par()’ command, which can be used to set several graphical parameters impacting the plot. Some of the available graphic parameters can be indicated in either a ‘par()’ or a ‘plot()’ command. The difference is that, once a parameter is set in a ‘par()’ command, it remains valid for subsequent interactions with the current graphic device, also influencing other graphic properties, while the use of the same command within a specific ‘plot()’ is restricted to that specific plot. For instance, consider the following command sequence:

```
x <- seq(0,10);    y <- x^2
plot(x,y,"b",col="blue")
```

This will plot a blue parabola of points joined by segments. However, the following code:

```
x <- seq(0,10);    y <- x^2
par(col="blue");   plot(x,y,"b)
```

will not only plot the line as blue, but also the lines of the surrounding frame. It will also influence subsequent plots performed on the currently active graphic device.

There are several parameters that can be used within the ‘plot()’ function. Some of the most frequently used include:

1. *xlab, ylab*: specify the label to be shown in the x and y-axes;
2. *xlim, ylim*: specify the interval to be considered along the x and y-axes;
3. *pch*: specifies the mark type;
4. *type*: specifies the type of curve, e.g. *type=“l”* for solid line;
5. *cex*: specifies the size of the marks;
6. *col*: indicates the color to be used;
7. *main*: defines the main title of the graph, to be shown above it;
8. *sub*: a subcaption, complementing the main caption;
9. *xaxt=”n”,yaxt=”n”*: indicate that the axes are not to be plotted;
10. *lwd*: specifies the width of lines;
11. *font*: specifies the characters font;
12. *bg*: sets the background color.

When using colors, keep in mind that R provides some built-in palettes generating functions, including ‘gray.colors’, ‘rainbow()’, ‘heat.colors()’, ‘topo.colors()’, terrain.colors()’ and ‘cm.colors()’, which yield a number of varying levels determined by the respective argument.

Each time a ‘plot()’ command is issued, it rewrites the previous one. In case the previous graphical output is to be preserved, so that the current one acts over it, the command ‘par(new=“TRUE”)’ can be used.

The margins of the plot region can be specified inside the ‘par()’ command through the directives ‘mai(bottom, left, top, right)’, for inner margins, and ‘omi(bottom, left, top, right)’ for outer margins. The values are given in inches, but ‘mar’ and ‘oma’ can be used for values in number of lines.

Once a plot has been drawn, we can add text onto it by using the command ‘text(x, y, ”text”)’. Several parameters, such as ‘col’, ‘cex’, etc., can be respectively applieed. The complete set of 657 built-in color names in R can be obtained by ‘colors()’.

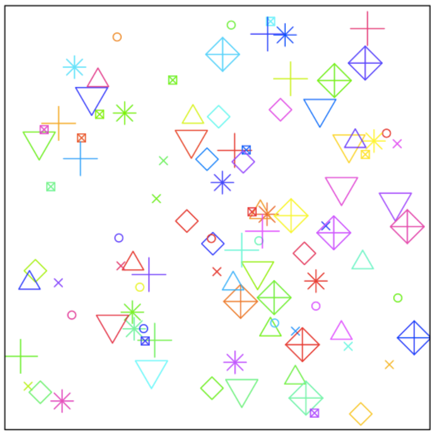
The following piece of code, whose output is depicted in Figure 3(a) illustrates some of the above mentioned features of the ‘plot()’ and ‘par()’ commands for drawing separated points.

```
rm(list = ls())
n <- 100;   hcols <- rainbow(n)
x <-runif(n,0,1);   y <-runif(n,0,1)
plot(x,y,col=hcols, xaxt="n",yaxt="n",xlab="",ylab="",
      pch=c(1,2,3,4,5,6,7,8,9), cex=c(1,2,3))
```

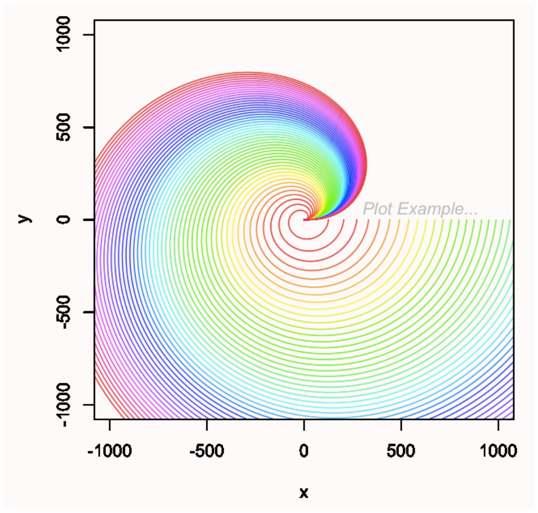
Observe the first line, used to clear the current R objects in case they are no longer needed.

The code below, with output shown in Figure 3(b) exemplifies some aspects of the ‘plot()’ and ‘par()’ commands for drawing joined points (curves).

```
rm(list = ls())
N <- 1000;   i <- seq(0,N);
```



(a)



(b)

Figure 3: Examples of the use of commands ‘plot()’ for plotting separated points (a) and curves (b). Observe that the spirals in (b) correspond to examples of the use of the command ‘plot()’ for drawing parametric curves (e.g. [6])

```
x <- y <- matrix(0,1,N)
n <- 50;   hcols <- rainbow(n)
par(mai = c(1, 1, 1, 1), bg = "snow")
for (k in 0:n){
  x <- ((i*k)^0.7)*cos(2*pi*i/N);
  y <- ((i*k)^0.7)*sin(2*pi*i/N)
  plot(x,y,col=hcols[k],type="l",
            xlim=c(-N,N), ylim=c(-N,N))
  par(new=TRUE)
}
text(600,50,"Plot Example...",col="gray",font=3)
```

4 Bar and Histogram Plots

Other types of 3D graphic output that are often useful include bar plots and histograms.

By making ‘plot(x,y,”h”)’, a plot in terms of narrow bars can be obtained. For instance:

```
N <- 200;   t <- seq(0,N-1);
x <-  matrix(0,1,N)
x <- cos(2*pi*t*3/N) + 0.3*cos(2*pi*t*5/N) + 2
plot(t,x,"h",main="Narrow bars")
```

Yields the graphic output shown in Figure 4

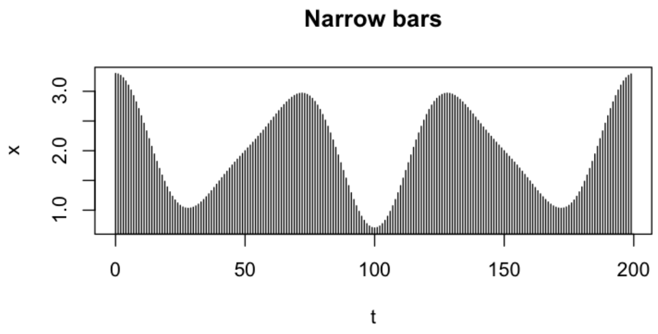


Figure 4: A narrow bar plot obtained by the plot command with type = ”h”.

Frequently, new types of graphs can be obtained by combining two or more graphic commands. For instance, if we have two subsequent plots, with respective types ‘s’ and ’S’, we obtain a ‘block’ graph:

```
x <- seq(-1,1,0.05);    y <- x^2
plot(x,y,"s",col="orange")
par(new=TRUE)
plot(x,y,"S",col="orange")
par(new=TRUE)
plot(x,y,"l")
```

whose output is given in Figure 5.

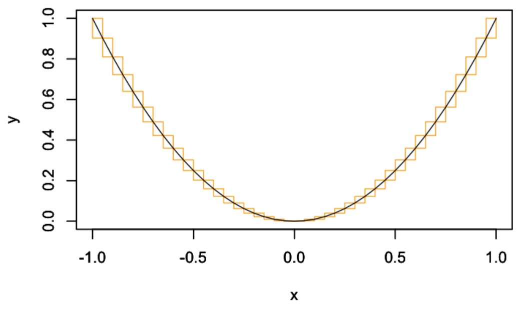


Figure 5: .

Another command that can be used for obtaining bar plots is ‘barplot(v)’, which shows the values in vector v as successive bars. Several parameters in this function can be used to control bars spacing, width, height, color, etc.

It is also possible to show more than one bar level, which is done by using a matrix, instead of a vector, as the argument. For instance,

```
barplot(t(Orange[1:7,2:3]),col=c("darkgreen","gold"))
legend("topleft", c("Tree age","Circumference"),
fill = c("darkgreen","gold") )
```

where 'orange' is a build-in dataset in R containing the age and circumference of orange trees, will lead to the graphical output in Figure 6.

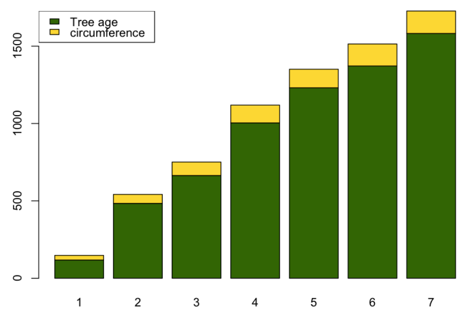


Figure 6: The 'barplot()' graphic function can be used to show more than one level of bars.

Another interesting way to visualize values in a vector is through the function 'dotchart()'. Now, the values are shown as points along a scale. Consider the following example:

```
dotchart(precip[1:10], bg = "blue")
```

where 'precipt' is a built-in vector in R containing precipitation in several American cities. Both values and city names are included in 'precip', as can be immediately verified by entering 'str(precip)'. The respective output can be seen in Figure 7.

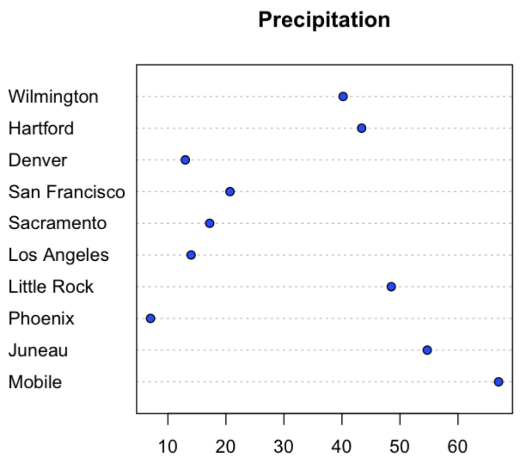


Figure 7: Dot-chart representation of part of the 'precip' R built-in dataset.

Histograms are another type of 2D graphical output that is often employed, especially in statistics. The 'hist()' command from R provides several facilities for plotting histograms of the data in a vector x, i.e. 'plot(x, breaks, plot, etc.)', where 'breaks' can be an integer number indicating the amount of histogram bars, a vector specifying the desired break-points, among other possibilities. When we make 'plot = FALSE' no graphical output is produced. In this case, the output can be directed to an external object, e.g. 'h ¡- hist(x, breaks=10, plot=FALSE)'.

Consider the following example:

```
x <- rnorm(300, 0, 1)
h <- hist(x,10,plot=FALSE)
plot(h, col="aquamarine")
```

where 'rnorm(300, 0, 1)' is a function used to obtain 300 normaly distributed points with mean 0 and standard deviation 1. The graphic ouput obtained by the plot command is shown in Figure 8.

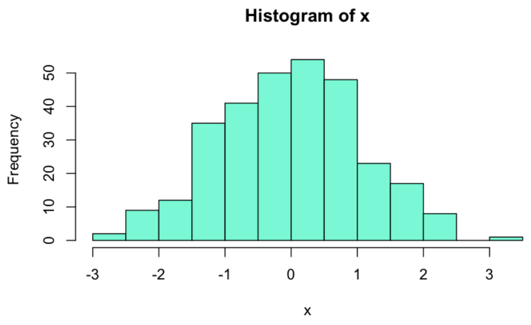


Figure 8: Histogram of 300 normally distributed values with zero means and unit variance obtained by combined use of the function 'hist()' and 'plot()'.

It is interesting to inspect the obtained 'h' object, by making 'str(h)'. We find that this object is a list of 6 components, including the breakpoints and respective counts, densities and bin midpoints. Therefore, the function 'hist()' used with 'plot=FALSE' provides a means for obtaining the several properties of histograms while avoiding respective graphical output.

5 3D Scatterplots

There are more than an interesting way to show points in 3D spaces in R, which can be accomplished by using different libraries (see Table 1). Let's start with the 'plot3d()' function, as employed in the following piece of code:

```
library(rgl)
x <- runif(10, 0, 1)
y <- runif(10, 0, 1)
z <- runif(10, 0, 1)
plot3d(x, y, z, size=7, col=c("red","green","blue"))
```

The first command loads the respective library, 'rgl', the 3 next commands generate vectors x, y and z with

10 random values uniformly distributed between 0 and 1. The last command produces the graphical output shown in Figure 9.

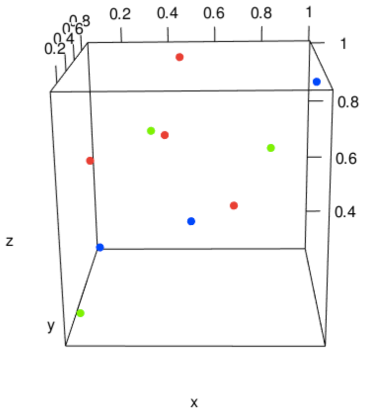


Figure 9: Graphical output obtained by using the 'plot3D()' function from the 'rgl' library. In several cases (it may depend on your operating system, configuration, etc.) the obtained graphic frame can be rotated in the R environment (the above picture is a snapshot) under control of the mouse.

One interesting feature of this function is that the obtained graphic output can be rotated interactively while dragging it with the mouse, enhancing the tridimensionality thanks to the parallax between points at different distances.

Another command that can be used to obtain 3D plots of points is the function 'scatter3D', from library 'plot3d', which is illustrated as follows:

```
library(plot3d)
x <- runif(10, 0, 1)
y <- runif(10, 0, 1)
z <- runif(10, 0, 1)
scatter3D(x, y, z, colkey=FALSE,type="s", bty = "g",
      lwd=8, col=c("orange","skyblue","green"),
      colvar=NULL, ticktype = "detailed")
text3D(x, y, z,  labels = as.character(seq(1,10)),
      colkey = FALSE, add= TRUE, cex=0.7)
```

Figure 10 illustrates the respectively obtained graphic result.

In the 'scatter3D()' function, the option 'colvar=FALSE' is used in order to avoid the color acting with respect to the z value, 'bty' sets the ruled box, 'lwd' controls the width of the lines used to draw the points (spheres), and 'ticktype = TRUE' implies the axes to be marked with a numeric scale rather than by arrows.

The scatter command is followed by a 'text3D()' function, which includes labels respectively to each plotted point, where the parameter 'cex' controls the size of the labels. The directive 'colkey = FALSE' used in both graphic commands avoids a palette legend to be included at the right-hand side of the plot.

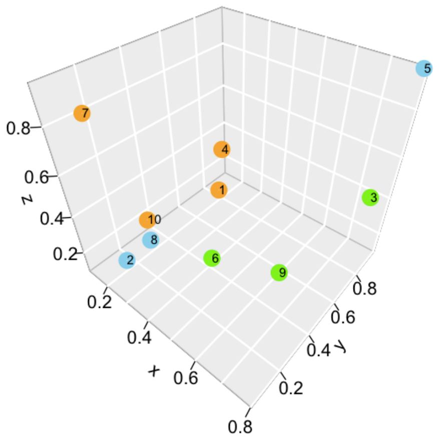


Figure 10: Graphic output for the 'scatter3D()' example.

6 Images

Images are particularly important in visualization, as they are produced in ever increasing quantity by the most diverse areas. Basically, a *gray-scale image* is a matrix where each element, a *pixel*, reflects the luminosity of a respective small region in the corresponding scene. Color images involve more matrices (typically 3 ou 4), such as in the RGB system.

There is a third type of color scheme that is used with some frequency, namely the so-called *false colors*, through the adoption of respective palettes.

An important point to be kept in mind while working with false-colors is that each of the obtained images is actually derived from a single matrix, therefore primarily being a gray-scale image. However, it is sometimes desired to assign a respective colors to each of the levels in the single matrix, which is implemented through a respective *palette* that implements this mapping. In R, palettes are limited to 1024 entries. It should be borne in mind that the choice of a suitable palette can be a challenging issue. For instance the color transitions in palettes may incorrectly emphasize secondary aspects in the imaged data, etc.

We have already used the concept of palette in the examples in Figure 3, where we adopted the 'rainbow' built-in function to generate respective palettes.

The basic command in R for plotting an image is 'image()'. It can take several arguments (please refer to the respective documentation), especially a matrix *im*, which corresponds to the respective gray-level image, i.e. 'plot(im)'. Figure 11 illustrates the result of 'image(volcano)', where volcano is the only image in the R built-in datasets, corresponding to measurements of topographical information on Auckland's Maunga Whau volcano. Actually, as it will be immediately revealed by entering 'str(volcano)', this is a numeric matrix with dimension 87×61, and by using the functions 'min()' and 'max()' we can learn that its entries range between 94 and 195.

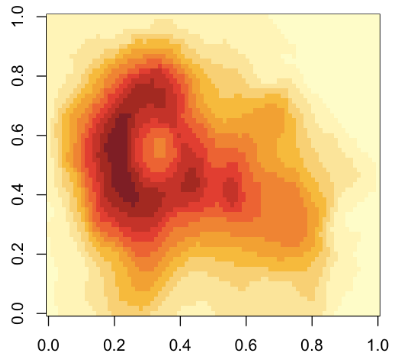


Figure 11: The output of the command 'image(volcano)', where 'volcano' is the image built-in dataset available in R.

Some interesting points can be observed from the graphic output in Figure 11. First, we observe that the x and y axes are measured in normalized fashion from 0 to 1, instead of from 1 to 87 and 1 to 61 (the respective image size), and that there are no axes labels or title. Second, we have that the x-axis corresponds to the row dimension of the matrix 'volcano', while the y-axis is associated to the columns. Third, we have that, though this image is inherently gray-level (a single matrix of values), a color output was obtained, which indicates that a false-color scheme based on some palette has been employed Interestingly, the default palette in the current environment has no effect on the 'image()' command. If a palette different from the default is to be used, we need to specify this as a parameter in the image command, i.e.:

```
hcols <- cm.colors(100)
image(volcano,col=hcols)
```

which yields the graphical output shown in Figure 12

The correspondence between the values i and j indexing the visualized matrix, e.g. $im[i,j]$, is as follows. The element $im[1,1]$ is shown at the bottom-left of the image, while the element $[length(im), length(im)]$ is shown at the top-right corner of the displayed image. This scheme is convenient because, when plotting functions and fields, we can have the indexing $im[ix,iy]$ to correspond to the more usual tuple representation (x,y), where ix and iy are respective indices of x and y.

However, it should be kept in mind that, when inspecting the matrix 'im' as lines in the R console, the rows and columns of the matrix will actually correspond to the columns and rows of the image, and in reverse order in the case of the former index.

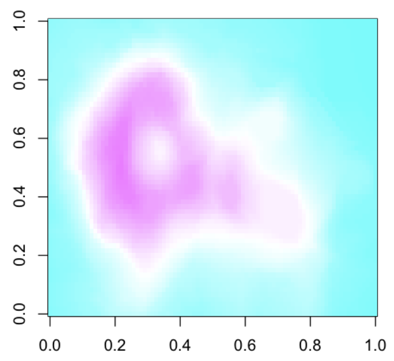


Figure 12: The volcano built-in dataset visualized with a palette generated by the 'cm.colors()' function.

7 Level Sets

Given a surface defined by a respective scalar field $\psi(x,y)$, its level-set at z is henceforth understood as the set of points (x,y,z) so that:

$$(x,y,z) \Leftrightarrow \psi(x,y) = z \tag{1}$$

The 'contour()' function can be used to obtain level sets of a scalar represented in terms of a respective matrix A, i.e. 'contour(A)'.

For instance, let's consider the scalar field $\psi(x,y) = xy$. We can have a respective matrix representation by making:

```
x <- y <- seq(-1,1,0.1)
A <- outer(x,y)
contour(A,xlab="x",ylab="y")
```

which yields the graphic output shown in Figure 13.

It is also possible to indicate to 'contour()' a parameter 'level' containing the z-values to be used for obtaining the respective level sets.

8 Creating Meshes

3D surfaces in several environments often rely on the concept of *meshes*. Basically, a surface in $\Re^3$ can be approached as a scalar field of the type $(x,y,f(x,y))$, with $x_{min} \leq x \leq x_{max}$ and $y_{min} \leq y \leq y_{max}$. Computationally, it is necessary to adopt respective resolutions Δx and Δy, so that $\Delta_x = (x_{max} - x_{min})/(N_x - 1)$ and $\Delta_y = (y_{max} - y_{min})/(N_y - 1)$, where N_x and N_y are the number of points along the x and y axes, respectively.

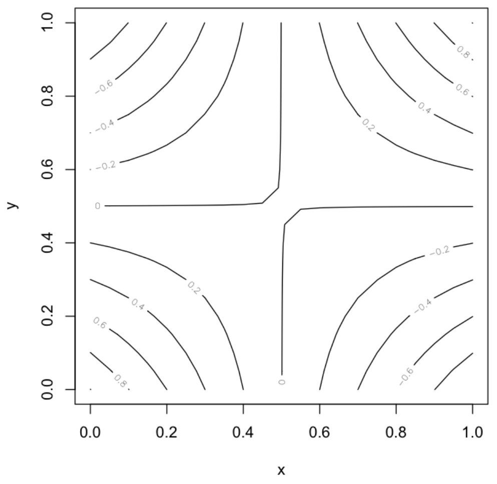


Figure 13: Example of contour plot of the scalar field $\psi(x, y) = xy$.

Then, the (x, y) domain can be sampled by respective indices, i.e. $x = (i_x - 1)\Delta x + x_{min}$ and $y = (i_y - 1)\Delta y + y_{min}$, with $i_x = 1, 2, \ldots, N_x$ and $i_y = 1, 2, \ldots, N_y$. Alternatively, we have that $i_x = (x - x_{min})\Delta_x + 1$ and $i_y = (y - y_{min})\Delta_y + 1$. Recall that the indices of lists, vectors, matrices and arrays in R start at '1', and not '0'.

As such, the instantiation of a surface depends on the evaluation of the respective function $f()$ over all permutations (actually, the Cartesian product) of considered x and y values. This is precisely when the concept of mesh enters the scene.

The command 'expand.grid(x,y)' can be used to obtain *a data frame* (a type of R data structure) containing the Cartesian product of x and y organized in pairwise fashion. We will use the 'mesh()' function in our surface and parametric surface examples. The command 'M <- mesh(x,y)', where x and y are vectors with respective lengths N_x and N_y yields a list of two elements,'M\$x' and 'M\$y', each corresponding to an $N_x \times N_y$ matrix. The former matrix contains all its columns identical to vector x, and the latter has all its rows identical to vector y.

Let's consider the following code:

```
library(plot3d)
x <- seq(1,5)
y <- seq(1,7)
M <- mesh(x,y)
```

The two matrices contained in the result 'M', i.e. 'M\$x' and 'M\$y' are:

```
> M$x
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    2    2    2    2    2    2    2
[3,]    3    3    3    3    3    3    3
[4,]    4    4    4    4    4    4    4
[5,]    5    5    5    5    5    5    5
```

and:

```
> M$y
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2    3    4    5    6    7
[2,]    1    2    3    4    5    6    7
[3,]    1    2    3    4    5    6    7
[4,]    1    2    3    4    5    6    7
[5,]    1    2    3    4    5    6    7
```

It is also possible to obtain the Cartesian product matrices directly by using matrix operations, e.g.:

```
v1x <- matrix(1,Nx,1)
v1y <- matrix(1,1,Ny)
xx <- x %*% v1y
yy <- v1x %*% y
```

9 Surfaces

A surface S in $\Re^3$ with domain in $\Re^2$ can be approached in terms of scalar fields of the type:

$$S : (x, y, \psi(x, y)) \tag{2}$$

For simplicity's sake, in the current work, we will use 'persp3D()' to plot this type of surface, though there are other commands such as 'surf3D()'. The latter function inputs the Cartesian products of 'x' and 'y', and the former input those vectors directly. In both cases the 'z' input is the matrix with the respective scalar field to be visualized.

The following piece of code defines a surface and visualizes it through 'persp3D()', with the graphical output being shown in Figure 14.

```
library(rgl)
library(plot3d)
psi <- function(x,y){ -(-x^2 - y^2) * (x+y) }

Nx <- 20;  Ny <- 10
xmin <- -2;  xmax <- 2;  dx <- (xmax-xmin)/(Nx-1)
ymin <- -1;  ymax <-1;   dy <- (ymax-ymin)/(Ny-1)
x <- seq(xmin,xmax,dx);  y <- seq(ymin,ymax,dy)
M <- mesh(x,y);   xx <- M$x;   yy <- M$ y

z <- psi(xx,yy)
persp3D(x = x, y = y, z = z, col="orange",
      xlim =c(xmin,xmax), ylim=c(xmin,xmax),
       colkey=FALSE, theta= -30,  alpha = 0.2,
       border = "darkgreen", ticktype="detailed",
      contour=list(levels = seq(-15,15)),
      xlab="x",ylab="y", zlab="psi(x,y)",
      cex.axis=0.6)
```

where the parameter 'alpha=0.2' controls the transparency of the rendered surface. The level-sets of the respectively visualized scalar field is also incorporated in this plot by including the directive 'contour=list(levels = seq(-15,15))' as argument of the 'persp3D()' function call. The parameter 'theta' controls the azimutal angle of the visualization, and the directive 'border' sets the color of the borders of the surface. No borders are shown if this directive is omitted.

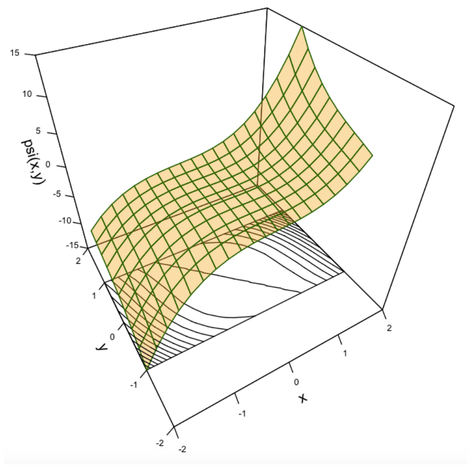


Figure 14: The surface $(x, y, \psi(x, y))$, with $\psi(x, y) = -(x^2+y^2)(x+y)$, as visualized by the respective program example.

The following program provides another example of surface visualization, with respective graphic output shown in Figure 15.

```
library(rgl)
library(plot3d)

psi <- function(x,y){ sin((x^2 + y^2)*2*pi/N*16)*
   exp(0.2*(-x^2 - y^2)) }

N <- 100;  xmin <- -3;  xmax <- 3
dx <- dy <- (xmax-xmin)/(N-1)
x <- y <-  seq(xmin,xmax,dx)
M <- mesh(x,y);  xx <- M$x;   yy <- M$ y

z <- psi(xx,yy)
hpall <- cm.colors(50)
persp3D(x = x, y = y, z = z, theta= -30,
    xlim =c(xmin,xmax), ylim=c(xmin,xmax),
      col=hpall, colkey=FALSE, border = "skyblue",
      alpha = 0.2, ticktype="detailed", lwd=0.08,
      xlab="x",ylab="y",zlab="psi(x,y)",cex.axis=0.6)
```

where the parameter 'lwd=0.08' controls the line width of the borders used in the surface rendering.

10 Parametric Surfaces

A parametric surface in 3D (i.e. the vector space $\Re^3$) can be understood as a scalar field of the type:

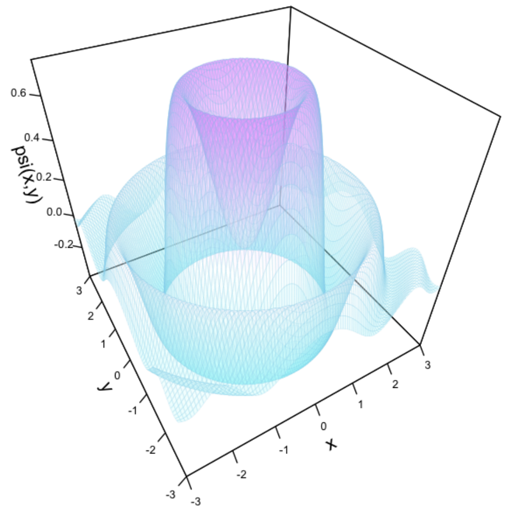


Figure 15: Another example of surface rendering by using the 'persp3D()' graphic function.

$$\psi(x, y, z) = \psi(\psi_x(u, v),\ \psi_y(u, v),\ \psi_z(u, v)) \qquad (3)$$

where $u, v \in \Re^2$ are the respective *parameters*, restricted to some interval, i.e. $v_{min} \leq v \leq v_{max}$ and $u_{min} \leq u \leq u_{max}$.

For instance, let's define a paraboloid that has its main axis coinciding with the reference versor $\hat{w} = \frac{\sqrt{3}}{3}(1, 1, 1)$. This vector defines a family of plans obeying the equation $x + y + z + c = 0$. For simplicity's sake, let's make $c = 0$, so that we have $x + y + z = 0$.

Now, we need to obtain two vectors $\vec{u}$ and $\vec{v}$ belonging to this plan, so as to define a respective orientation frame. To determine $\vec{u}$, we can make $\vec{U} = (x = 1, y = 0)$ and find its image in the plan $x + y + z = 0$, which implies $z = -x$ and $\vec{U} = (1, 0, -1)$. As we need a respective versor, we impose:

$$\hat{u} = \frac{\vec{U}}{||\vec{U}||} = 1 \Rightarrow \frac{(1, 0, -1)}{||(1, 0, -1)||} = \frac{\sqrt{2}}{2}(1, 0, -1) \qquad (4)$$

Now, we need to find a second vector $\vec{V} = (x_V, y_V, z_V)$ which obeys $\vec{V} \cdot \hat{u} = 0$ and $\vec{V} \cdot \hat{w} = 0$, hence:

$$\vec{V} \cdot \hat{u} = 0 \Rightarrow \frac{\sqrt{2}}{2}(x_V - z_v) = 0 \Rightarrow x_V = z_V = x_o$$

$$\vec{V} \cdot \hat{w} = 0 \Rightarrow \frac{\sqrt{2}}{2}(x_V + y_V + z_V) = 0 \Rightarrow$$
$$\Rightarrow (2x_o + y_V) = 0 \Rightarrow y_V = -2x_o$$

implying that $\vec{V} = (x_o, -2x_o + x_o)$. So, the respective versor $\hat{v}$ can now be obtained as:

$$||\vec{V}|| = 1 \Rightarrow ||(x_o, -2x_o, x_o)|| = 1 \Rightarrow$$
$$\Rightarrow x_o^2 + (2x_o)^2 + x_o^2 = 1 \Rightarrow$$
$$\Rightarrow 6x_o^2 = 1 \Rightarrow x_o = \pm\frac{\sqrt{6}}{6} \qquad (5)$$

In order to obtain a right-hand orientation for the system defined by $\hat{u}$, $\hat{v}$, and $\hat{w}$, we adopt $x_o = -\frac{\sqrt{6}}{6}$, so that we have $\hat{v} = \frac{\sqrt{6}}{6}(-1, 2, -1)$.

So we have the following correspondence:

$$\begin{bmatrix} \hat{u} \\ \hat{v} \\ \hat{w} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & -\frac{\sqrt{6}}{6} \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix}$$

The above relationship directly leads to the following variable transformation:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & -\frac{\sqrt{6}}{6} \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

The plan $x + y + z = 0$ can now be expressed parametrically as $(\sqrt{2}/2(x-z), \sqrt{6}/6(-x+2y-z), \sqrt{3}/3(x+y+z))$. The following piece of code can be used to visualize this plan as depicted in Figure 16.

```
library(rgl)
library(plot3d)
a <- 1;
umin <- -a;  umax <- a;  du <- 0.1
vmin <- -a;  vmax <- a;  dv <- 0.1
us <- seq(umin,umax,du)
vs <- seq(vmin,vmax,dv)
M <- mesh(us,vs);  u <- M$x;  v <- M$y

c1 <- sqrt(2)/2;   c2 <- sqrt(6)/6; c3 <- sqrt(3)/3
x <- c1*u - c2*v;  y <- 2*c2*v;     z <- -c1*u -c2*v

# ================================
# plot plan
persp3D(x = x, y = y, z = z, xlim =c(xmin,xmax),
    ylim=c(xmin,xmax),  col="skyblue", colkey=FALSE,
    theta= 35, alpha = 0.2, ticktype="detailed",
    xlab="x", ylab="y", zlab="psi(x,y)",
    bty = "g", cex.axis=0.6)
points3D(0,0,0,add=TRUE, col="red", pch=16, cex=1.5)
# plot point and arrowrs
arrows3D(c(0,0,0),c(0,0,0),c(0,0,0),
   c(-c2,c3,c1), c(2*c2,c3,0), c(-c2,c3,-c1),
   add=TRUE, col=c("darkblue", "red", "darkgreen"),
   lwd=4)
# ================================
```

where 'bty' is a parameter setting the type of the box (please refer to the command documentation).

From Equation 6 we obtain:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & -\frac{\sqrt{6}}{6} \\ \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

As this matrix is orthogonal by construction (we are performing a rotation), the respective inverse is given by

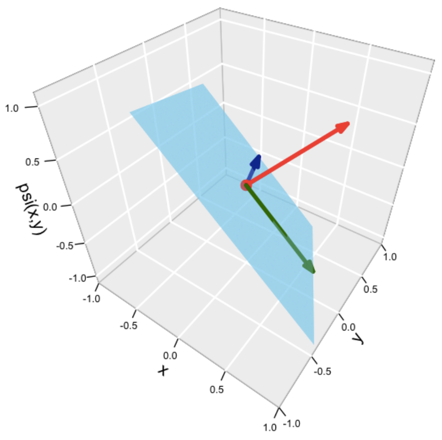


Figure 16: Visualization of the plan $x + y + z = 0$ expressed as $(\sqrt{2}/2(x-z), \sqrt{6}/6(-x+2y-z), \sqrt{3}/3(x+y+z))$. The versors $\hat{u}$, $\hat{v}$, and $\hat{w}$ are shown respectively in green, blue and red.

its transpose, and we get:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} \\ 0 & \frac{\sqrt{6}}{3} & \frac{\sqrt{3}}{3} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

Now, we are ready to define our tilted paraboloid as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} \\ 0 & \frac{\sqrt{6}}{3} & \frac{\sqrt{3}}{3} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} u \\ v \\ u^2 + v^2 \end{bmatrix}$$

The tilted paraboloid can now be visualized by replacing the following portion by the respectively marked section in the previous program example. Figure 17 illustrates the respectively obtained graphical output.

```
# ================================
# plot plan
persp3D(x = x, y = y, z = z, xlim =c(xmin,xmax),
    ylim=c(xmin,xmax),  col="blue", colkey=FALSE,
    theta= 35, border = "royalblue3", alpha = 0.1,
    ticktype="detailed", bty = "g", cex.axis=0.6)
# plot tilted paraboloid
persp3D(x = xx, y = yy, z = zz, xlim =c(xmin,xmax),
    ylim=c(xmin,xmax),  col=hcols, colkey=FALSE,
    theta= -30, border = "darkolivegreen3",
    alpha = 0.1,  ticktype="detailed", xlab="x",
    ylab="y", zlab="z",cex.axis=0.6, add=TRUE)
# ================================
```

Another example of visualization, now of a torus, is provided in the following piece of code, with respectively obtained result shown in Figure 18.

```
r <- 1;   R <- 2;  a <- seq(0, 2*pi, pi/20)
A <- mesh(a,a);  u <- A$x;  v <- A$y
```

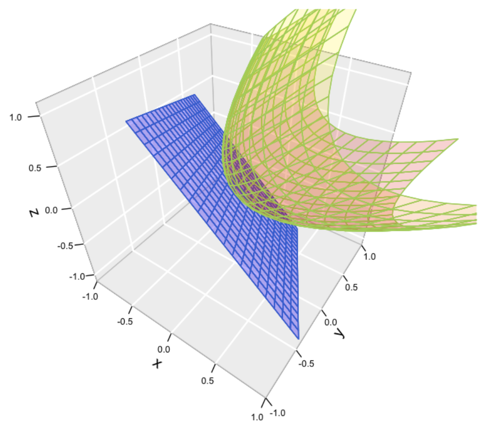


Figure 17: The tilted paraboloid example.

```
x = (R + r*cos(v)) * cos(u);
y = (R + r*cos(v)) * sin(u)
z = r * sin(v)

persp3D(x, y, z,  colkey = FALSE,
      xlim=c(-3,3), ylim=c(-3,3), zlim=c(-3,3),
      ticktype = "detailed", cex.axis = 0.5,
      col = "lightgreen", border="brown",
      bty="b2",  alpha=0.3)
```

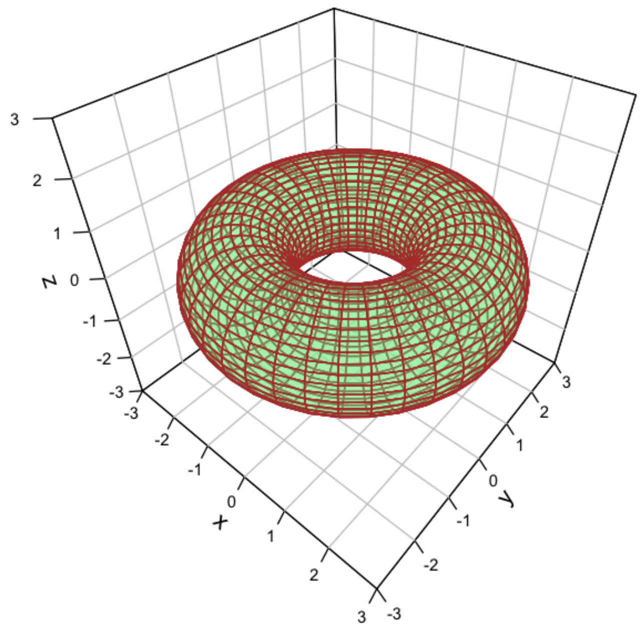


Figure 18: Visualization of a torus parametric surface.

11 Vector Fields

Vector fields can also be visualized in R by using different approaches. In the present work, we consider the command 'quiver()', which takes the 'x' and 'y' coordinates where the arrows are to be shown, together with the respective cmomponents 'fx(x,y)' and 'fy(x,y)' of the respective vector field $\vec{\phi}(\phi_x(x,y), \phi_y(x,y))$, yielding a respective visualization, as illustrated in the following piece of code, whose output is shown in Figure 19.

```
rm(list = ls())
library(pracma)
library(plot3D)

ffx <- function(x,y) -x ; ffy <- function(x,y) -y

xmin <- -1;  xmax <- 1;   dx <- 0.1;
xs <- seq(xmin,xmax,dx);  Nx <- length(xs)
A <- mesh(xs,xs);   x <- A$x;  y <- A$y
a <- 0.1;   fx <- a * ffx(x,y);   fy <- a *  ffy(x,y)

# set white canvas
plot(c(xmin,xmax),c(xmin,xmax),col="white",
     xlab="x",ylab="y",col.lab="blue")
# plot vector field
quiver(x, y, fx, fy, code=2, col="darkgreen",
     scale=1,xlab="",ylab="", lwd=1.5)
```

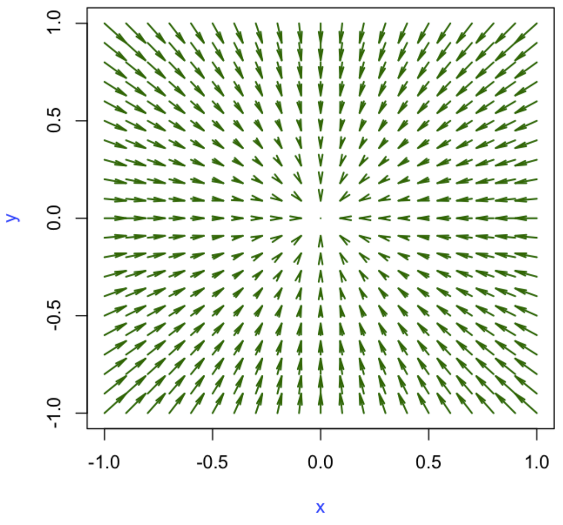


Figure 19: Visualization of the vector field $\phi(-x, -y)$.

12 Concluding Remarks

Computer graphics and scientific visualization are motivating areas in science and technology for their intrinsic approaches as well as for the virtually unlimited applications.

In the present work, we aimed at providing a concise introduction, with respective examples, to a few of the many graphic resources and functionalities in the R environment. There are much more to know about R visualization. It is hoped that the reader will be motivated to probe further.

Acknowledgments.
Luciano da F. Costa thanks CNPq (grant no. 307085/2018-0) for sponsorship. This work has benefited from FAPESP grant 15/22308-2.

References

[1] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics Principles and Practice in C.* Addison-Wesley, 1995.

[2] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufmann, 2016.

[3] D. Solomon. *The computer graphics manual.* Springer Verlag, 2011.

[4] L. da F. Costa. CDT series. Researchgate, 2020. `https://www.researchgate.net/project/Costas-Didactic-Texts-CDTs`. [Online; accessed 18-Apr-2020.].

[5] L. da F. Costa. An abridged introduction to R. Researchgate, 2020. `https://www.researchgate.net/publication/340771350_An_Abridged_Introduction_to_R_CDT-29`. [Online; accessed 18-Apr-2020.].

[6] L. da F. Costa. A mosaic of multivariate calculus. Researchgate, 2020. `https://www.researchgate.net/publication/340570370_A_Mosaic_of_Multivariate_Calculus_CDT-27`. [Online; accessed 18-Apr-2020.].

[7] rgl library. R Documentation, 2020. `https://www.rdocumentation.org/packages/rgl/versions/0.100.50`. Online; accessed 25-March-2020.

[8] plot3d library. R Documentation, 2020. `https://www.rdocumentation.org/packages/plot3D/versions/1.3`. Online; accessed 25-March-2020.

[9] Graphics library. R Documentation, 2020. `https://www.rdocumentation.org/packages/graphics/versions/3.6.2`. Online; accessed 25-March-2020.

[10] pracma library. R Documentation, 2020. `https://www.rdocumentation.org/packages/pracma/versions/1.9.9`. Online; accessed 25-March-2020.